dmo_anl_vw_tot_mov_periodo

October 26, 2024

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import os
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
import uuid
from datetime import datetime
from dotenv import load_dotenv
import boto3
from skimpy import generate_test_data, skim

# Configurações de visualização
%matplotlib inline
sns.set(style="whitegrid")

"""
Importa todas as bibliotecas necessárias para a análise exploratória, incluindo␣
 ↪ferramentas para
redução de dimensionalidade e clustering.

Bibliotecas:
- pandas: Manipulação de dados
- numpy: Operações numéricas
- matplotlib: Visualização de dados
- seaborn: Visualização estatística
- os: Interação com o sistema operacional
- sklearn.decomposition.PCA: Análise de Componentes Principais
- sklearn.cluster.KMeans: Algoritmo de Clustering KMeans
- sklearn.preprocessing.StandardScaler: Escalonamento de dados
- boto3: Interação com AWS S3
"""
```

[1]:
```
'\nImporta todas as bibliotecas necessárias para a análise exploratória,
incluindo ferramentas para\nredução de dimensionalidade e
clustering.\n\nBibliotecas:\n- pandas: Manipulação de dados\n- numpy: Operações
numéricas\n- matplotlib: Visualização de dados\n- seaborn: Visualização
estatística\n- os: Interação com o sistema operacional\n-
sklearn.decomposition.PCA: Análise de Componentes Principais\n-
sklearn.cluster.KMeans: Algoritmo de Clustering KMeans\n-
sklearn.preprocessing.StandardScaler: Escalonamento de dados\n- boto3: Interação
com AWS S3\n'
```

[2]:
```
# Definir caminho do arquivo CSV
DATA_DIR = r'C:\Users\Inteli\Downloads\dados_cptm\caixa_preta'
csv_file = 'dmo_anl_vw_tot_mov_periodo.csv'

# Caminho completo para o arquivo CSV
csv_path = os.path.join(DATA_DIR, csv_file)

# Verificar se o arquivo CSV existe e carregá-lo
df = None  # Inicializar a variável df para evitar problemas caso o arquivo não␣
 ↪exista
if os.path.exists(csv_path):
    # Carregar o arquivo CSV em um DataFrame
    df = pd.read_csv(csv_path)
    print("Arquivo CSV carregado com sucesso.")
else:
    print(f"Arquivo {csv_file} não encontrado no diretório {DATA_DIR}.")
```

```
Arquivo CSV carregado com sucesso.
```

[3]:
```
df.head()
```

[3]:

	id_dt_hora_minuto	cod_bilh	cd_estac_bu	dt_validacao	total_validacoes	tipo_dia
0	46	3001	619	10/24/23 00:00:00	59	U
1	46	3001	619	10/25/23 00:00:00	107	U
2	46	3001	619	10/26/23 00:00:00	67	U
3	46	3001	619	10/27/23 00:00:00	66	U
4	46	3001	619	10/28/23 00:00:00	78	S

[4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 3028393 entries, 0 to 3028392
Data columns (total 6 columns):
 #   Column             Dtype 
---  ------             ----- 
 0   id_dt_hora_minuto  int64 
 1   cod_bilh           int64 
 2   cd_estac_bu        int64 
 3   dt_validacao       object
 4   total_validacoes   int64 
 5   tipo_dia           object
dtypes: int64(4), object(2)
memory usage: 138.6+ MB
```

[5]:
```
df.describe()
```

[5]:

	id_dt_hora_minuto	cod_bilh	cd_estac_bu	total_validacoes
count	3.028393e+06	3.028393e+06	3.028393e+06	3.028393e+06
mean	5.379721e+01	2.281298e+04	6.300346e+02	1.247573e+01
std	2.206573e+01	3.160558e+04	8.872497e+01	3.239844e+01
min	1.000000e+00	2.530000e+03	0.000000e+00	1.000000e+00
25%	3.400000e+01	3.000000e+03	5.590000e+02	1.000000e+00
50%	5.400000e+01	5.001000e+03	6.160000e+02	4.000000e+00
75%	7.200000e+01	2.500000e+04	7.070000e+02	1.100000e+01
max	9.600000e+01	9.800000e+04	9.120000e+02	1.894000e+03

[6]:
```
df.isnull().sum()
```

[6]:
```
id_dt_hora_minuto    0
cod_bilh             0
cd_estac_bu          0
dt_validacao         0
total_validacoes     0
tipo_dia             0
dtype: int64
```

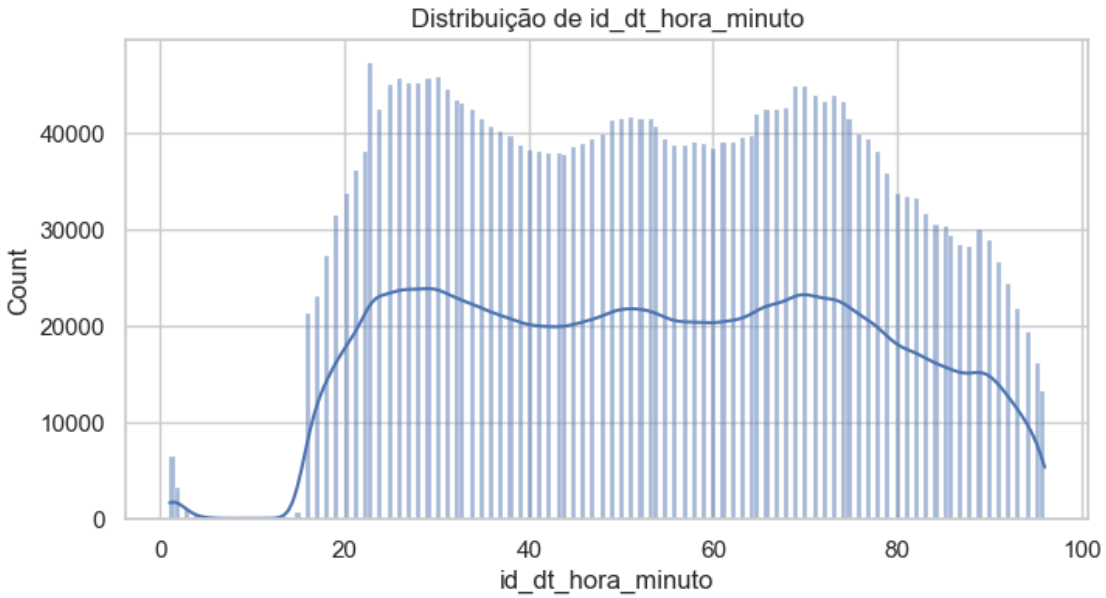
[7]:
```
print(f"--- Estatísticas Descritivas de {df} ---")
display(df.describe(include='all'))
print("\\n")
```

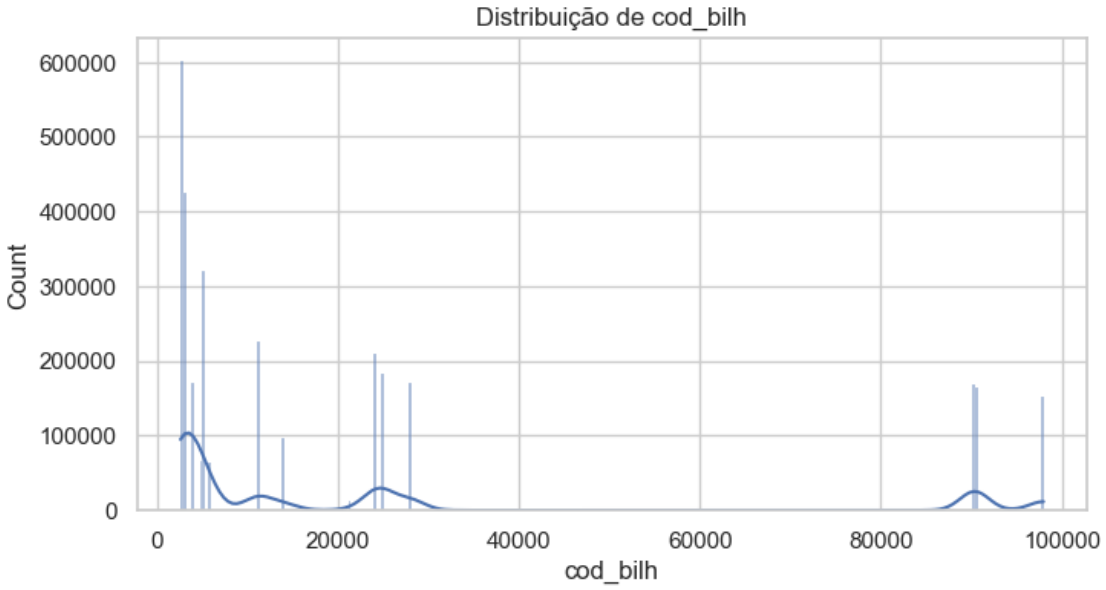
```
--- Estatísticas Descritivas de          id_dt_hora_minuto  cod_bilh
cd_estac_bu       dt_validacao  \
0                        46      3001          619  10/24/23 00:00:00
1                        46      3001          619  10/25/23 00:00:00
2                        46      3001          619  10/26/23 00:00:00
3                        46      3001          619  10/27/23 00:00:00
4                        46      3001          619  10/28/23 00:00:00
…                       …       …             …                 …
3028388                  46      3001          619  10/19/23 00:00:00
```

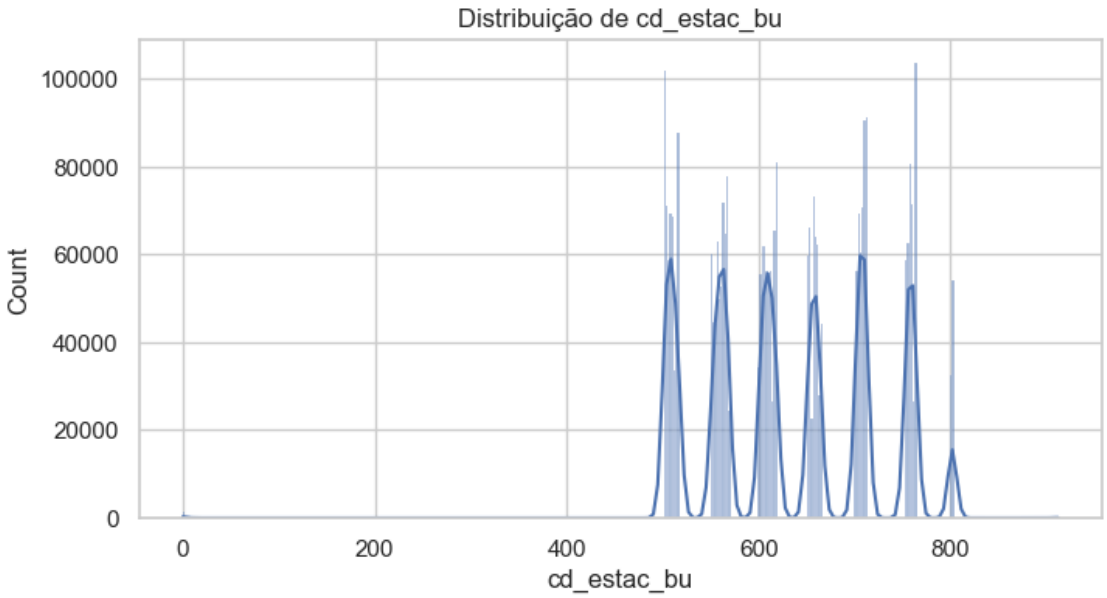
```
3028389                 46      3001          619  10/20/23 00:00:00
3028390                 46      3001          619  10/21/23 00:00:00
3028391                 46      3001          619  10/22/23 00:00:00
3028392                 46      3001          619  10/23/23 00:00:00

         total_validacoes tipo_dia
0                      59        U
1                     107        U
2                      67        U
3                      66        U
4                      78        S
…                     …       …
3028388                55        U
3028389                49        U
3028390                71        S
3028391                22        D
3028392                48        U

[3028393 rows x 6 columns] ---

        id_dt_hora_minuto      cod_bilh   cd_estac_bu       dt_validacao  \
count        3.028393e+06  3.028393e+06  3.028393e+06            3028393
unique                NaN           NaN           NaN                 31
top                   NaN           NaN           NaN  10/11/23 00:00:00
freq                  NaN           NaN           NaN             111225
mean         5.379721e+01  2.281298e+04  6.300346e+02                NaN
std          2.206573e+01  3.160558e+04  8.872497e+01                NaN
min          1.000000e+00  2.530000e+03  0.000000e+00                NaN
25%          3.400000e+01  3.000000e+03  5.590000e+02                NaN
50%          5.400000e+01  5.001000e+03  6.160000e+02                NaN
75%          7.200000e+01  2.500000e+04  7.070000e+02                NaN
max          9.600000e+01  9.800000e+04  9.120000e+02                NaN

        total_validacoes tipo_dia
count       3.028393e+06  3028393
unique               NaN        3
top                  NaN        U
freq                 NaN  2148168
mean        1.247573e+01      NaN
std         3.239844e+01      NaN
min         1.000000e+00      NaN
25%         1.000000e+00      NaN
50%         4.000000e+00      NaN
75%         1.100000e+01      NaN
max         1.894000e+03      NaN

\n
```

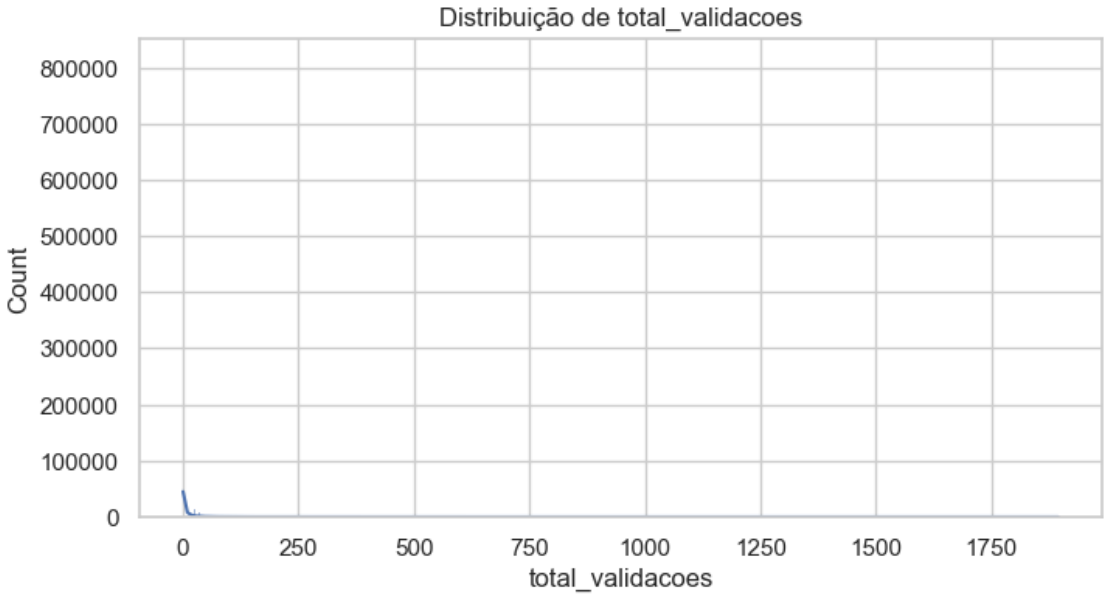
[8]:
```
# Exibir as distribuições das variáveis numéricas no DataFrame df
print(f"--- Distribuições de Variáveis no DataFrame ---")
for column in df.select_dtypes(include=[np.number]).columns:
    plt.figure(figsize=(8, 4))
    sns.histplot(df[column].dropna(), kde=True)
    plt.title(f'Distribuição de {column}')
    plt.show()
```

```
--- Distribuições de Variáveis no DataFrame ---
```



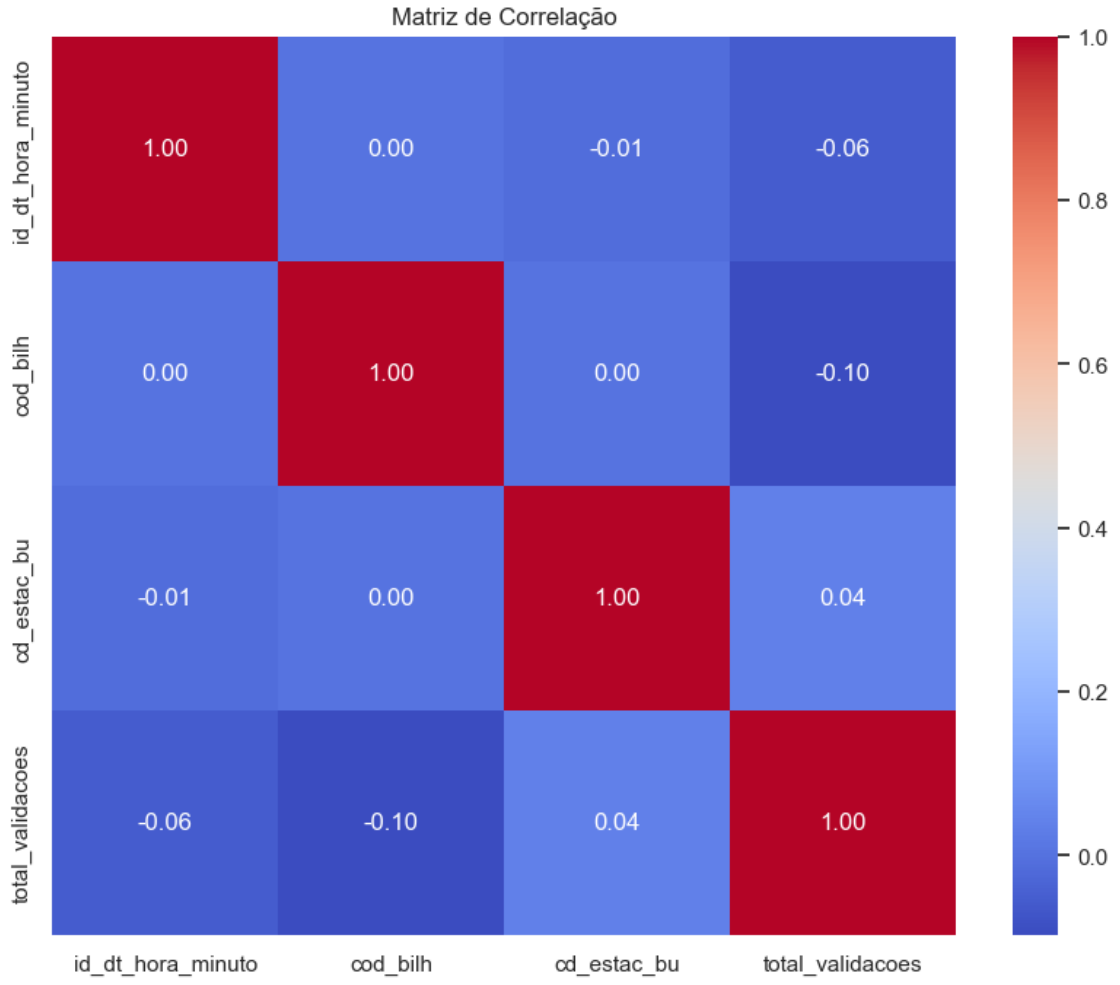






[9]:
```
# Selecionar apenas colunas numéricas para calcular a matriz de correlação
numeric_df = df.select_dtypes(include=[np.number])
```

```
# Verificar se há colunas numéricas suficientes para gerar a matriz de␣
 ↪correlação
if numeric_df.shape[1] > 1:
    plt.figure(figsize=(10, 8))
    corr = numeric_df.corr()
    sns.heatmap(corr, annot=True, fmt=".2f", cmap='coolwarm')
    plt.title('Matriz de Correlação')
    plt.show()
else:
    print("Não há colunas numéricas suficientes para calcular a correlação.")
```



[10]:
```
def analisar_dados():
    """
    Faz uma análise resumida do DataFrame.

    Usa a função `skim()` para exibir um resumo com estatísticas básicas
```

```
    e informações sobre os dados do DataFrame.

    Exemplo:
        skim(df)
    """
    skim(df)

analisar_dados()
```

```
                        skimpy summary
         Data Summary                  Data Types                                 ␣
↪                                                                                 ␣
↪
  dataframe           Values      Column Type   Count                             ␣
↪
                                                                                  ␣
↪
  Number of rows      3028393     int32         4                                 ␣
↪
  Number of columns   6           string        2                                 ␣
↪
                                                                                  ␣
↪
                                                number                            ␣
↪

  column_name             NA    NA %    mean      sd        p0      p25     p50    ␣
↪p75      p100     hist

  id_dt_hora_minuto        0       0    53.8     22.07       1       34      54   ␣
↪    72       96
  cod_bilh                 0       0   22810     31610    2530     3000    5001   ␣
↪ 25000    98000
  cd_estac_bu              0       0     630     88.72       0      559     616   ␣
↪   707      912
  total_validacoes         0       0   12.48      32.4       1        1       4   ␣
↪    11     1894

                                                string                            ␣
↪

  column_name                 NA       NA %         words per row                 ␣
↪ total words

  dt_validacao                 0          0                                      2␣
↪                  6056786
```

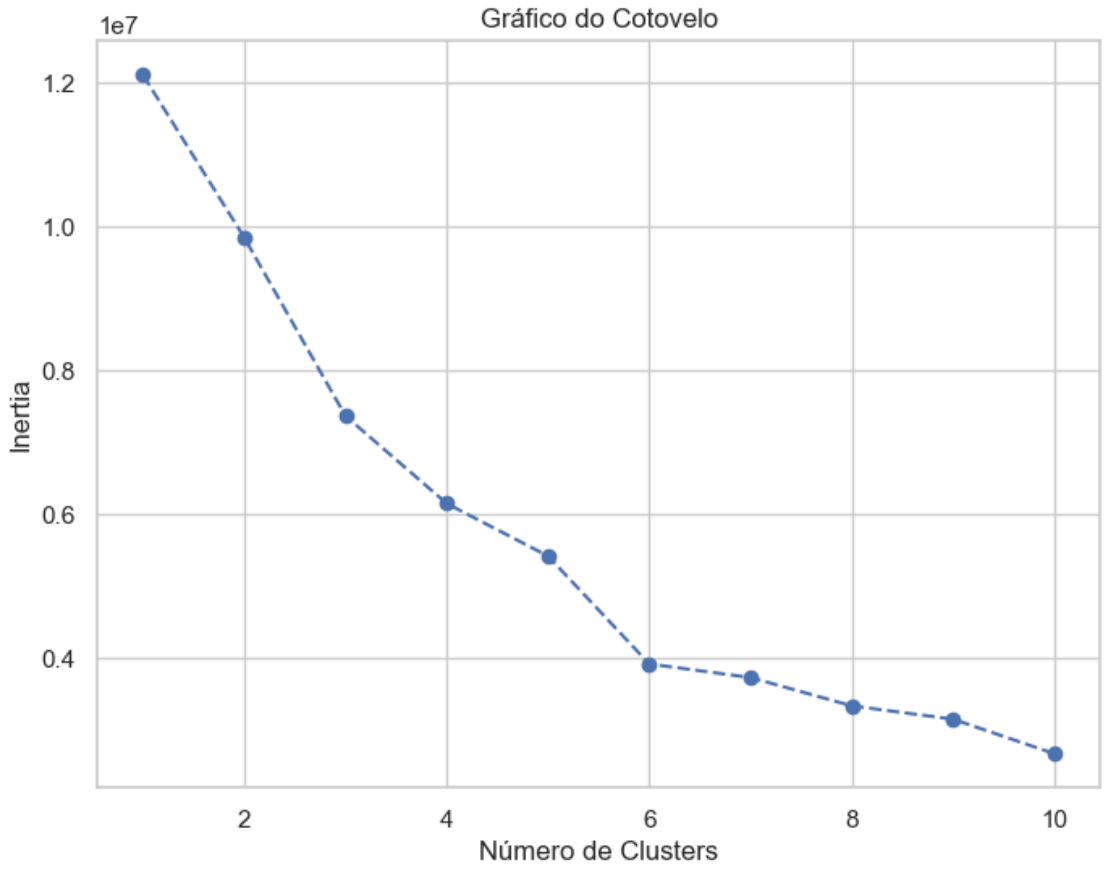
```
  tipo_dia                        0             0                          1␣
↪                  3028393
```

End

[11]:
```
def realizar_pca(df, n_componentes=2):
    """
    Realiza a Análise de Componentes Principais (PCA) para reduzir a␣
↪dimensionalidade dos dados.

    Args:
        df (pd.DataFrame): DataFrame com os dados a serem analisados.
        n_componentes (int): Número de componentes principais a serem mantidos.

    Returns:
        pd.DataFrame: DataFrame com as componentes principais.
        PCA: Objeto PCA treinado.
    """
    # Selecionar apenas variáveis numéricas e remover valores ausentes
    numeric_df = df.select_dtypes(include=[np.number]).dropna()

    # Escalonar os dados
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(numeric_df)

    # Aplicar PCA
    pca = PCA(n_components=n_componentes)
    principal_components = pca.fit_transform(scaled_data)

    # Criar DataFrame das componentes principais
    pca_df = pd.DataFrame(data=principal_components,
                          columns=[f'PC{i+1}' for i in range(n_componentes)])

    return pca_df, pca

# Aplicar PCA no DataFrame df
pca_df, pca_modelo = realizar_pca(df)

# Exibir os resultados
print(f"PCA realizado no DataFrame:")
print(f"Variância explicada por componente: {pca_modelo.
↪explained_variance_ratio_}\n")
```

```
PCA realizado no DataFrame:
Variância explicada por componente: [0.28028363 0.2525598 ]
```

[12]:
```python
def plot_grafico_cotovelo(df, max_clusters=10):
    """
    Plota o gráfico do cotovelo para identificar o número ideal de clusters.

    Args:
        df (pd.DataFrame): DataFrame com os dados a serem analisados.
        max_clusters (int): Número máximo de clusters a serem testados.

    Returns:
        None: Exibe o gráfico do cotovelo.
    """
    # Selecionar apenas variáveis numéricas e remover valores ausentes
    numeric_df = df.select_dtypes(include=[np.number]).dropna()

    # Escalonar os dados
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(numeric_df)

    # Lista para armazenar a soma das distâncias quadradas dentro dos clusters␣
 ↪(inertia)
    inertias = []

    # Aplicar KMeans com diferentes números de clusters
    for k in range(1, max_clusters + 1):
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(scaled_data)
        inertias.append(kmeans.inertia_)

    # Plotar o gráfico do cotovelo
    plt.figure(figsize=(8, 6))
    plt.plot(range(1, max_clusters + 1), inertias, marker='o', linestyle='--')
    plt.title('Gráfico do Cotovelo')
    plt.xlabel('Número de Clusters')
    plt.ylabel('Inertia')
    plt.show()

# Aplicar o gráfico do cotovelo no DataFrame df
plot_grafico_cotovelo(df, max_clusters=10)
```



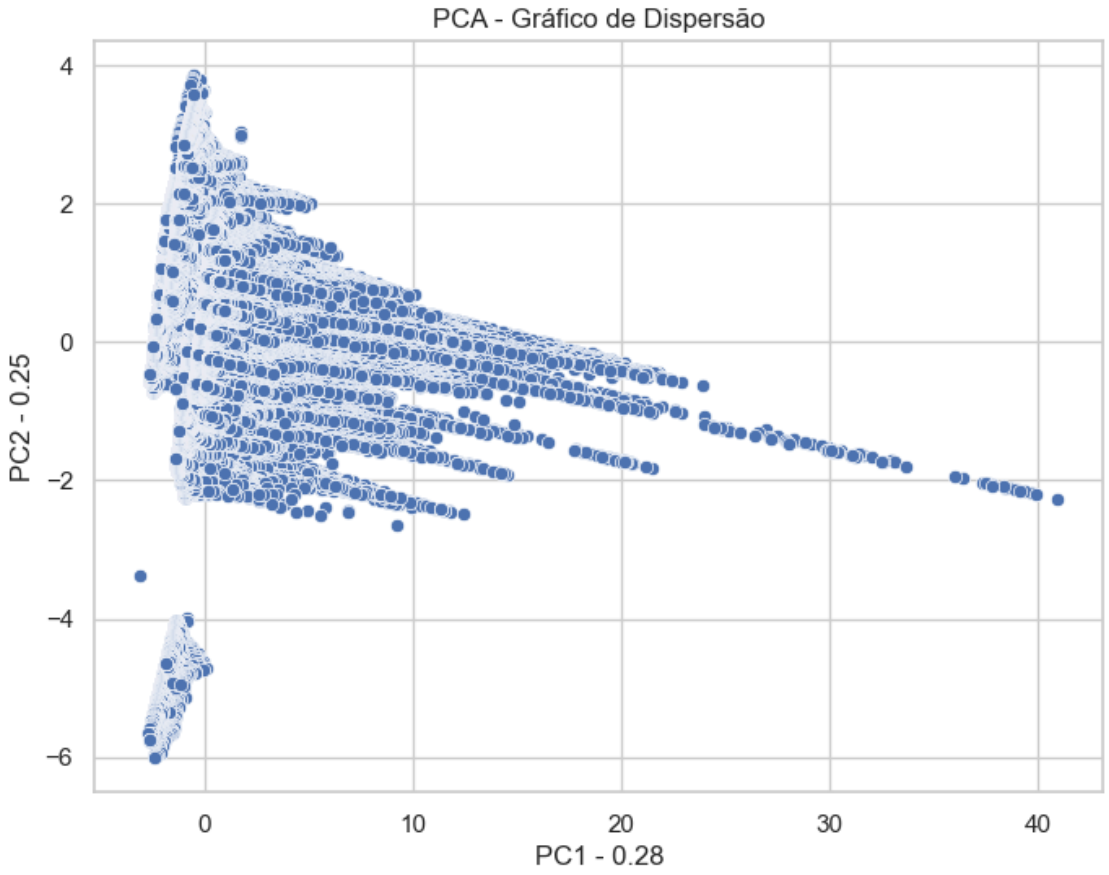
[13]:
```python
def realizar_kmeans(df, n_clusters=5):
    """
    Aplica o algoritmo de clustering KMeans para agrupar os dados de um␣
 ↪DataFrame.

    Args:
        df (pd.DataFrame): DataFrame com os dados a serem agrupados.
        n_clusters (int): Número de clusters a serem formados (default = 3).

    Returns:
        KMeans: Modelo KMeans treinado.
        np.ndarray: Rótulos dos clusters para cada amostra.
    """
    # Selecionar apenas variáveis numéricas e remover valores ausentes
    numeric_df = df.select_dtypes(include=[np.number]).dropna()

    # Escalonar os dados
    scaler = StandardScaler()
    scaled_data = scaler.fit_transform(numeric_df)
```

```
    # Aplicar KMeans
    kmeans = KMeans(n_clusters=n_clusters, random_state=42)
    kmeans.fit(scaled_data)
    labels = kmeans.labels_

    return kmeans, labels, scaled_data

# Aplicar KMeans no DataFrame df
kmeans_modelo, labels, scaled_data = realizar_kmeans(df, n_clusters=5)
```

[14]:
```
def plot_kmeans_clusters(df, kmeans, labels, scaled_data):
    """
    Cria um gráfico interativo em 2D ou 3D usando Plotly, mostrando os clusters e os centroides.

    Args:
        df (pd.DataFrame): DataFrame original.
        kmeans (KMeans): Modelo KMeans treinado.
        labels (np.ndarray): Rótulos dos clusters para cada amostra.
        scaled_data (np.ndarray): Dados escalonados usados no KMeans.

    Returns:
        None: Exibe o gráfico interativo com Plotly.
    """
    # Reduzir a dimensionalidade para 2D com PCA (caso tenha mais de 2 dimensões)
    if scaled_data.shape[1] > 2:
        from sklearn.decomposition import PCA
        pca = PCA(n_components=2)
        reduced_data = pca.fit_transform(scaled_data)
    else:
        reduced_data = scaled_data

    # Criar DataFrame com os componentes principais ou dados originais
    plot_df = pd.DataFrame(reduced_data, columns=['PC1', 'PC2'])
    plot_df['cluster'] = labels

    # Adicionar os centroides ao gráfico
    centroids = kmeans.cluster_centers_
    if scaled_data.shape[1] > 2:
        centroids = pca.transform(centroids)

    # Gráfico de dispersão com Plotly
    fig = px.scatter(
        plot_df, x='PC1', y='PC2', color=plot_df['cluster'].astype(str),
        title="KMeans Clustering com Centroides",
```

```
        labels={'color': 'Cluster'}
    )

    # Adicionar os centroides ao gráfico
    for i, centroid in enumerate(centroids):
        fig.add_scatter(
            x=[centroid[0]], y=[centroid[1]],
            mode='markers',
            marker=dict(size=12, symbol='x', color='red'),
            name=f'Centroid {i+1}'
        )

    fig.show()


# Exibir o gráfico interativo
plot_kmeans_clusters(df, kmeans_modelo, labels, scaled_data)
```

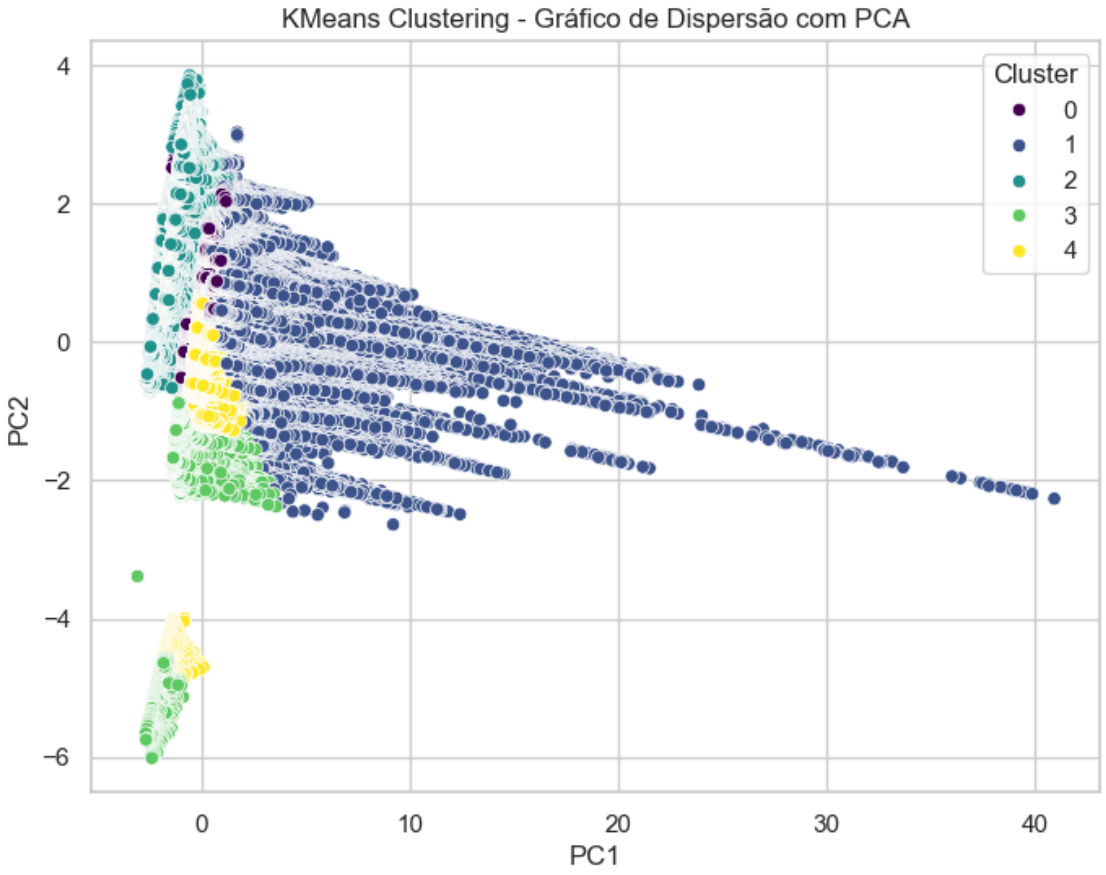
[15]:
```
def plot_pca_scatter(pca_df, pca_modelo):
    """
    Cria um gráfico de dispersão das duas primeiras componentes principais␣
 ↪resultantes da PCA.

    Args:
        pca_df (pd.DataFrame): DataFrame com os dados das componentes␣
 ↪principais resultantes da PCA (PC1, PC2).
        pca_modelo (PCA): O modelo PCA treinado, contendo as variâncias␣
 ↪explicadas para as componentes principais.

    Returns:
        None: Exibe o gráfico de dispersão das componentes principais (PC1 e␣
 ↪PC2).
    """
    # Criar o gráfico de dispersão
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x='PC1', y='PC2', data=pca_df)

    # Adicionar título e rótulos aos eixos com a variância explicada por cada␣
 ↪componente
    plt.title('PCA - Gráfico de Dispersão')
    plt.xlabel(f'PC1 - {pca_modelo.explained_variance_ratio_[0]:.2f}')
    plt.ylabel(f'PC2 - {pca_modelo.explained_variance_ratio_[1]:.2f}')

    # Exibir o gráfico
    plt.show()

plot_pca_scatter(pca_df, pca_modelo)
```



[16]:
```python
def plot_kmeans_clusters(df, labels, pca_modelo):
    """
    Cria um gráfico de dispersão das duas primeiras componentes principais␣
↪resultantes da PCA,
    colorido pelos rótulos de clusters gerados pelo KMeans.

    Args:
        df (pd.DataFrame): DataFrame original com os dados a serem agrupados.
        labels (np.ndarray): Rótulos dos clusters gerados pelo KMeans.
        pca_modelo (PCA): O modelo PCA já ajustado aos dados para reduzir a␣
↪dimensionalidade.

    Returns:
        None: Exibe o gráfico de dispersão com as componentes principais e os␣
↪clusters.
    """
    # Reduzir a dimensionalidade para 2D usando PCA para visualização
    pca_df, _ = realizar_pca(df, n_componentes=2)
```

```
    # Criar o gráfico de dispersão, colorindo pelos rótulos de cluster
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x='PC1', y='PC2', hue=labels, palette='viridis',␣
 ↪data=pca_df)

    # Adicionar título e rótulos aos eixos
    plt.title('KMeans Clustering - Gráfico de Dispersão com PCA')
    plt.xlabel('PC1')
    plt.ylabel('PC2')
    plt.legend(title='Cluster')

    # Exibir o gráfico
    plt.show()

plot_kmeans_clusters(df, labels, pca_modelo)

"""
Cria um gráfico de dispersão das duas primeiras componentes principais␣
 ↪resultantes da PCA,
colorido pelos rótulos de clusters gerados pelo KMeans.

Funcionalidade:
- Reduz a dimensionalidade dos dados usando PCA para projetar em 2D.
- Plota as componentes principais (PC1 e PC2) e colore os pontos de acordo com␣
 ↪os clusters gerados pelo KMeans.
- Os clusters são visualizados em um gráfico de dispersão para análise visual.

Args:
    df (pd.DataFrame): DataFrame original com os dados a serem agrupados.
    labels (np.ndarray): Rótulos dos clusters gerados pelo KMeans.
    pca_modelo (PCA): O modelo PCA já ajustado aos dados para reduzir a␣
 ↪dimensionalidade.

Returns:
    None: O gráfico de dispersão é exibido com os clusters identificados pelo␣
 ↪KMeans.
"""
```

```
c:\Users\Inteli\Documents\2024-2B-T10-SI08-G05\env\Lib\site-
packages\IPython\core\pylabtools.py:170: UserWarning:

Creating legend with loc="best" can be slow with large amounts of data.
```



[16]:
```
'\nCria um gráfico de dispersão das duas primeiras componentes principais
resultantes da PCA,\ncolorido pelos rótulos de clusters gerados pelo
KMeans.\n\nFuncionalidade:\n- Reduz a dimensionalidade dos dados usando PCA para
projetar em 2D.\n- Plota as componentes principais (PC1 e PC2) e colore os
pontos de acordo com os clusters gerados pelo KMeans.\n- Os clusters são
visualizados em um gráfico de dispersão para análise visual.\n\nArgs:\n    df
(pd.DataFrame): DataFrame original com os dados a serem agrupados.\n    labels
(np.ndarray): Rótulos dos clusters gerados pelo KMeans.\n    pca_modelo (PCA): O
modelo PCA já ajustado aos dados para reduzir a dimensionalidade.\n\nReturns:\n
None: O gráfico de dispersão é exibido com os clusters identificados pelo
KMeans.\n'
```

0.0.1 Resumo das Descobertas

- **Descoberta 1**: A distribuição de `dt_hora_minuto` confirma a existência de horários de pico no sistema de transporte.
- **Descoberta 2**: A distribuição de `cod_bilh` indica que o sistema tem uma variedade de bilhetes para os passageiros.
- **Descoberta 3**: A distribuição de `cd_estac_bu` mostra que a demanda por transporte público

varia significativamente entre as diferentes estações.

- **Descoberta 4**: A distribuição de `total_validacoes` sugere que eventos especiais podem ter um impacto significativo no volume de passageiros.

0.0.2 Hipóteses

1. **Sazonalidade**: Os dados de embarque podem ter padrões sazonais, com tipos específicos de bilhetes ou categorias de usuários que aumentam em determinados períodos. Esta hipótese pode ser investigada com uma análise temporal mais detalhada.
2. **Padrão de Fluxo**: Pode ser utilizada para analisar o fluxo de passageiros ao longo do tempo, identificando horários de pico, dias da semana com maior movimento, etc.
3. **Segmentação de Usuários**: Talvez, com dados adicionais, poderia ser utilizada para segmentar os usuários por tipo de bilhete, horário de uso ou estação de origem/destino, pois alguns grupos têm maior tendência a validar bilhetes em determinados dias da semana (finais de semana vs dias úteis).